1. Business Understanding

```
# Loading packages
library(ggplot2)
library(corrplot)
library(caret)
library(randomForest)
library(pROC)
library(ggcorrplot)
library(dplyr)
library(tidyverse)
library(ggpubr)
library(ggbiplot)
library(devtools)
library(factoextra)
```

2. Data Understanding and Preparation

The dataset contains maternal records of 1019 female patients. The patients consisted of 109 women aged 18 years and 19 years, and their ages range from 20 years old to 38 years old. In 50.39% (514 patients) the women had positive results (pathologic) and had previous health issues that put them at risk. In reason 0 or 1 of Risk Score Assessment (RSA) classification of the stages of fetal health.

Source: https://www.kaggle.com/datasets/andrewmvd/maternal-health-risk-data

```
# Loading the dataset
mydata <- read.csv("/Maternal Health Risk Data Set.csv")
head(mydata)
```

```
  age systolic_blood_pressure diastolic_blood_pressure blood_glucose body_temperature heart_rate risk_level
1  25                    130                        80       15.00             98.0       86     high risk
2  35                    140                        90       13.00             98.0       70     high risk
3  29                    90                         70        8.00             100.0      80      high risk
4  30                    140                        85       7.00             98.0       70      high risk
5  35                    120                        60       6.1              98.0       76      low risk
6  23                    140                        80       7.01             98.0       70      high risk
```

```
# Checking the dimension
dim(mydata)
```

```
[1] 1014    7
```

```
# Checking the structure
str(mydata)
```

```
'data.frame':   1014 obs. of  7 variables:
 $ age                  : int  25 35 29 30 35 23 32 19 29 35 ...
 $ systolic_blood_pressure: int  130 140 90 140 120 140 90 120 120 120 ...
 $ diastolic_blood_pressure: int  80 90 70 85 60 80 60 90 90 70 ...
 $ blood_glucose        : num  15 13 8 7 6.1 7.01 6.9 7.5 7.01 7.5 ...
 $ body_temperature     : num  98 98 100 98 98 98 98 98 98 98 ...
 $ heart_rate           : int  86 70 80 70 76 70 70 77 70 70 ...
 $ risk_level           : chr  "high risk" "high risk" "high risk" "high risk" ...
```

```
# Checking the summary
summary(mydata)
```

```
      age             systolic_blood_pressure diastolic_blood_pressure blood_glucose body_temperature   heart_rate
 Min.   :10.00   Min.   :  70.0   Min.   :  49.0   Min.   : 6.00   Min.   : 98.00   Min.   : 7.00
 1st Qu.:19.00   1st Qu.: 100.0   1st Qu.:  65.0   1st Qu.: 6.90   1st Qu.: 98.00   1st Qu.:70.00
 Median :26.00   Median : 120.0   Median :  75.0   Median : 7.50   Median : 98.00   Median :76.00
 Mean   :29.871  Mean   : 113.2   Mean   :  76.46  Mean   : 8.725  Mean   : 98.665  Mean   :74.301
 3rd Qu.:39.00   3rd Qu.: 120.0   3rd Qu.:  80.0   3rd Qu.: 7.90   3rd Qu.: 98.00   3rd Qu.:80.00
 Max.   :70.00   Max.   : 160.0   Max.   : 100.0   Max.   :19.00   Max.   :103.00   Max.   :90.00
   risk_level
 Length:1014
 Class :character
 Mode  :character
```

```
# Checking the unique values
unique(mydata$age)
unique(mydata$systolic_blood_pressure)
unique(mydata$diastolic_blood_pressure)
unique(mydata$blood_glucose)
unique(mydata$body_temperature)
unique(mydata$heart_rate)
unique(mydata$risk_level)
```

```
 [1] 25 35 29 30 23 32 19 20 38 37 ...
 [1] 130 140 90 120 100 160 70 110 150
 [1] 80 90 70 60 85 75 49 65 100
 [1] 15.00 13.00 8.00 7.00 6.1 7.01 6.9 7.5 6.00 8.00 ...
 [1] 98.0 100.0 98.00 103.00
 [1] 86 70 80 77 76 90
 [1] "high risk" "low risk"
```

Preparing a data dictionary for ease of understanding (1)

```
# Creating a data dictionary for the variables
Variable <- c("age", "systolic_blood_pressure", "diastolic_blood_pressure", "blood_glucose", "body_temperature", "heart_rate", "risk_level")
Description <- c("Age of the patient", "Blood pressure measurement", "Diastolic blood pressure measurement", "Blood glucose level", "Body temperature", "Heart rate", "Risk level classification")
Type <- c("Numerical", "Numerical", "Numerical", "Numerical", "Numerical", "Numerical", "Categorical")
data_dictionary <- data.frame(Variable, Description, Type)
print(data_dictionary)
```

```
  Variable                                   Description        Type
1      age                                Age of the patient   Numerical
2 systolic_blood_pressure   Blood pressure measurement   Numerical
3 diastolic_blood_pressure   Diastolic blood pressure measurement   Numerical
4   blood_glucose                          Blood glucose level   Numerical
5 body_temperature                          Body temperature   Numerical
6       heart_rate                                 Heart rate   Numerical
7       risk_level                       Risk level classification   Categorical
```

```
# Checking the data dictionary
View(data_dictionary)
```

```
# Checking for missing values
colSums(is.na(mydata))
```

```
                age systolic_blood_pressure diastolic_blood_pressure          blood_glucose     body_temperature
                  0                       0                       0                      0                    0
         heart_rate             risk_level
                  0                       0
```

```
# Checking for duplicates
sum(duplicated(mydata))
```

```
[1] 0
```

```
# Checking the distribution of the target variable
table(mydata$risk_level)
prop.table(table(mydata$risk_level))*100
```

```
high risk  low risk
  0.5138   0.4862
```

Numerical Features

```
# Selecting numerical variables
num_vars <- c("age", "systolic_blood_pressure", "diastolic_blood_pressure", "blood_glucose", "body_temperature", "heart_rate")
cor_matrix <- cor(mydata[, num_vars])
print(cor_matrix)
```

```
                        age systolic_blood_pressure diastolic_blood_pressure blood_glucose body_temperature   heart_rate
age               1.000000               0.3811468               0.2708460    0.4556673    -0.15526115 -0.07101930
systolic_blood_pressure   0.381146               1.0000000               0.7952321    0.3908109    -0.29080284  0.012710942
diastolic_blood_pressure  0.270846               0.7952321               1.0000000    0.2673503    -0.26949558  0.019393897
blood_glucose             0.455667               0.3908109               0.2673503    1.0000000    -0.04656964  0.393115338
body_temperature         -0.155261              -0.2908028              -0.2694956   -0.0465696     1.00000000 -0.199571603
heart_rate               -0.071019               0.0127109               0.0193939    0.3931153    -0.19957160  1.000000000
```

```
# Plotting correlation matrix
library(corrplot)
corrplot(cor_matrix, method = "circle", type = "upper", tl.cex = 0.8, tl.col = "black")
```

```
# Boxplots for numerical variables
par(mfrow = c(2, 3))
boxplot(mydata$age, main = "age")
boxplot(mydata$systolic_blood_pressure, main = "systolic_blood_pressure")
boxplot(mydata$diastolic_blood_pressure, main = "diastolic_blood_pressure")
boxplot(mydata$blood_glucose, main = "blood_glucose")
boxplot(mydata$body_temperature, main = "body_temperature")
boxplot(mydata$heart_rate, main = "heart_rate")
```

```
# Boxplots for numerical variables by risk_level
par(mfrow = c(2, 3))
boxplot(age ~ risk_level, main = "age")
boxplot(systolic_blood_pressure ~ risk_level, main = "systolic_blood_pressure")
boxplot(diastolic_blood_pressure ~ risk_level, main = "diastolic_blood_pressure")
boxplot(blood_glucose ~ risk_level, main = "blood_glucose")
boxplot(body_temperature ~ risk_level, main = "body_temperature")
boxplot(heart_rate ~ risk_level, main = "heart_rate")
```

```
# Checking for outliers
boxplot.stats(mydata$age)$out
boxplot.stats(mydata$systolic_blood_pressure)$out
boxplot.stats(mydata$diastolic_blood_pressure)$out
boxplot.stats(mydata$blood_glucose)$out
boxplot.stats(mydata$body_temperature)$out
boxplot.stats(mydata$heart_rate)$out
```

```
numeric(0)
numeric(0)
numeric(0)
numeric(0)
numeric(0)
numeric(0)
```

Categorical Features

```
# Selecting categorical variables
table(mydata$risk_level)
prop.table(table(mydata$risk_level))*100
```

```
high risk  low risk
  0.5138   0.4862
```

```
# Bar plot for risk_level
ggplot(mydata, aes(x = risk_level))
  geom_bar(fill = "steelblue")
  ggtitle("Distribution of Risk Level")
  xlab("Risk Level")
  ylab("Count")
  theme_minimal()
```



```
# Bar plot for risk_level by age
ggplot(mydata, aes(x = age, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Age")
  xlab("Age")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by systolic_blood_pressure
ggplot(mydata, aes(x = systolic_blood_pressure, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Systolic Blood Pressure")
  xlab("Systolic Blood Pressure")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by diastolic_blood_pressure
ggplot(mydata, aes(x = diastolic_blood_pressure, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Diastolic Blood Pressure")
  xlab("Diastolic Blood Pressure")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by blood_glucose
ggplot(mydata, aes(x = blood_glucose, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Blood Glucose")
  xlab("Blood Glucose")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by body_temperature
ggplot(mydata, aes(x = body_temperature, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Body Temperature")
  xlab("Body Temperature")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by heart_rate
ggplot(mydata, aes(x = heart_rate, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Heart Rate")
  xlab("Heart Rate")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Checking for duplicates
sum(duplicated(mydata))
```

```
[1] 0
```

```
# Checking the distribution of the target variable
table(mydata$risk_level)
prop.table(table(mydata$risk_level))*100
```

```
high risk  low risk
  0.5138   0.4862
```

```
# Bar plot for risk_level
ggplot(mydata, aes(x = risk_level))
  geom_bar(fill = "steelblue")
  ggtitle("Distribution of Risk Level")
  xlab("Risk Level")
  ylab("Count")
  theme_minimal()
```



```
# Bar plot for risk_level by age
ggplot(mydata, aes(x = age, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Age")
  xlab("Age")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by systolic_blood_pressure
ggplot(mydata, aes(x = systolic_blood_pressure, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Systolic Blood Pressure")
  xlab("Systolic Blood Pressure")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by diastolic_blood_pressure
ggplot(mydata, aes(x = diastolic_blood_pressure, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Diastolic Blood Pressure")
  xlab("Diastolic Blood Pressure")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by blood_glucose
ggplot(mydata, aes(x = blood_glucose, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Blood Glucose")
  xlab("Blood Glucose")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by body_temperature
ggplot(mydata, aes(x = body_temperature, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Body Temperature")
  xlab("Body Temperature")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Bar plot for risk_level by heart_rate
ggplot(mydata, aes(x = heart_rate, fill = risk_level))
  geom_bar(position = "dodge")
  ggtitle("Risk Level by Heart Rate")
  xlab("Heart Rate")
  ylab("Count")
  fill("Risk Level")
  theme_minimal()
```



```
# Checking for outliers using IQR method
outliers <- boxplot.stats(mydata$age)$out
length(outliers)
```

```
[1] 0
```

2.1.2 Bivariate Analysis

Numerical Features

Correlation

```
# Correlation matrix
cor_matrix <- cor(mydata[, c("age", "systolic_blood_pressure", "diastolic_blood_pressure", "blood_glucose", "body_temperature", "heart_rate")])
print(cor_matrix)
```

```
                        age systolic_blood_pressure diastolic_blood_pressure blood_glucose body_temperature   heart_rate
age               1.000000               0.3811468               0.2708460    0.4556673    -0.15526115 -0.07101930
systolic_blood_pressure   0.381146               1.0000000               0.7952321    0.3908109    -0.29080284  0.012710942
diastolic_blood_pressure  0.270846               0.7952321               1.0000000    0.2673503    -0.26949558  0.019393897
blood_glucose             0.455667               0.3908109               0.2673503    1.0000000    -0.04656964  0.393115338
body_temperature         -0.155261              -0.2908028              -0.2694956   -0.0465696     1.00000000 -0.199571603
heart_rate               -0.071019               0.0127109               0.0193939    0.3931153    -0.19957160  1.000000000
```

```
# Visualizing correlation matrix
library(ggcorrplot)
ggcorrplot(cor_matrix, hc.order = TRUE, type = "lower", lab = TRUE, lab_size = 3, method = "circle", colors = c("tomato2", "white", "steelblue"), title = "Correlation Matrix", ggtheme = theme_minimal())
```


This matrix indicated variables can not highly correlated. This is good as it indicates absence of multicollinearity.

Analysis with target variable

Our objective is to predict Death Event. Analysis of the features with target variable could give us insights into the effect they could have on our predictions.

```
# Boxplots for numerical variables by risk_level
par(mfrow = c(2, 3))
boxplot(age ~ risk_level, main = "age")
boxplot(systolic_blood_pressure ~ risk_level, main = "systolic_blood_pressure")
boxplot(diastolic_blood_pressure ~ risk_level, main = "diastolic_blood_pressure")
boxplot(blood_glucose ~ risk_level, main = "blood_glucose")
boxplot(body_temperature ~ risk_level, main = "body_temperature")
boxplot(heart_rate ~ risk_level, main = "heart_rate")
```



```
# Checking for outliers using IQR method
outliers <- boxplot.stats(mydata$age)$out
length(outliers)
```

```
[1] 0
```

2.1.3 Bivariate Analysis

Numerical Features

Correlation

```
# Correlation matrix
cor_matrix <- cor(mydata[, c("age", "systolic_blood_pressure", "diastolic_blood_pressure", "blood_glucose", "body_temperature", "heart_rate")])
print(cor_matrix)
```

```
                        age systolic_blood_pressure diastolic_blood_pressure blood_glucose body_temperature   heart_rate
age               1.000000               0.3811468               0.2708460    0.4556673    -0.15526115 -0.07101930
systolic_blood_pressure   0.381146               1.0000000               0.7952321    0.3908109    -0.29080284  0.012710942
diastolic_blood_pressure  0.270846               0.7952321               1.0000000    0.2673503    -0.26949558  0.019393897
blood_glucose             0.455667               0.3908109               0.2673503    1.0000000    -0.04656964  0.393115338
body_temperature         -0.155261              -0.2908028              -0.2694956   -0.0465696     1.00000000 -0.199571603
heart_rate               -0.071019               0.0127109               0.0193939    0.3931153    -0.19957160  1.000000000
```

```
# Visualizing correlation matrix
library(ggcorrplot)
ggcorrplot(cor_matrix, hc.order = TRUE, type = "lower", lab = TRUE, lab_size = 3, method = "circle", colors = c("tomato2", "white", "steelblue"), title = "Correlation Matrix", ggtheme = theme_minimal())
```

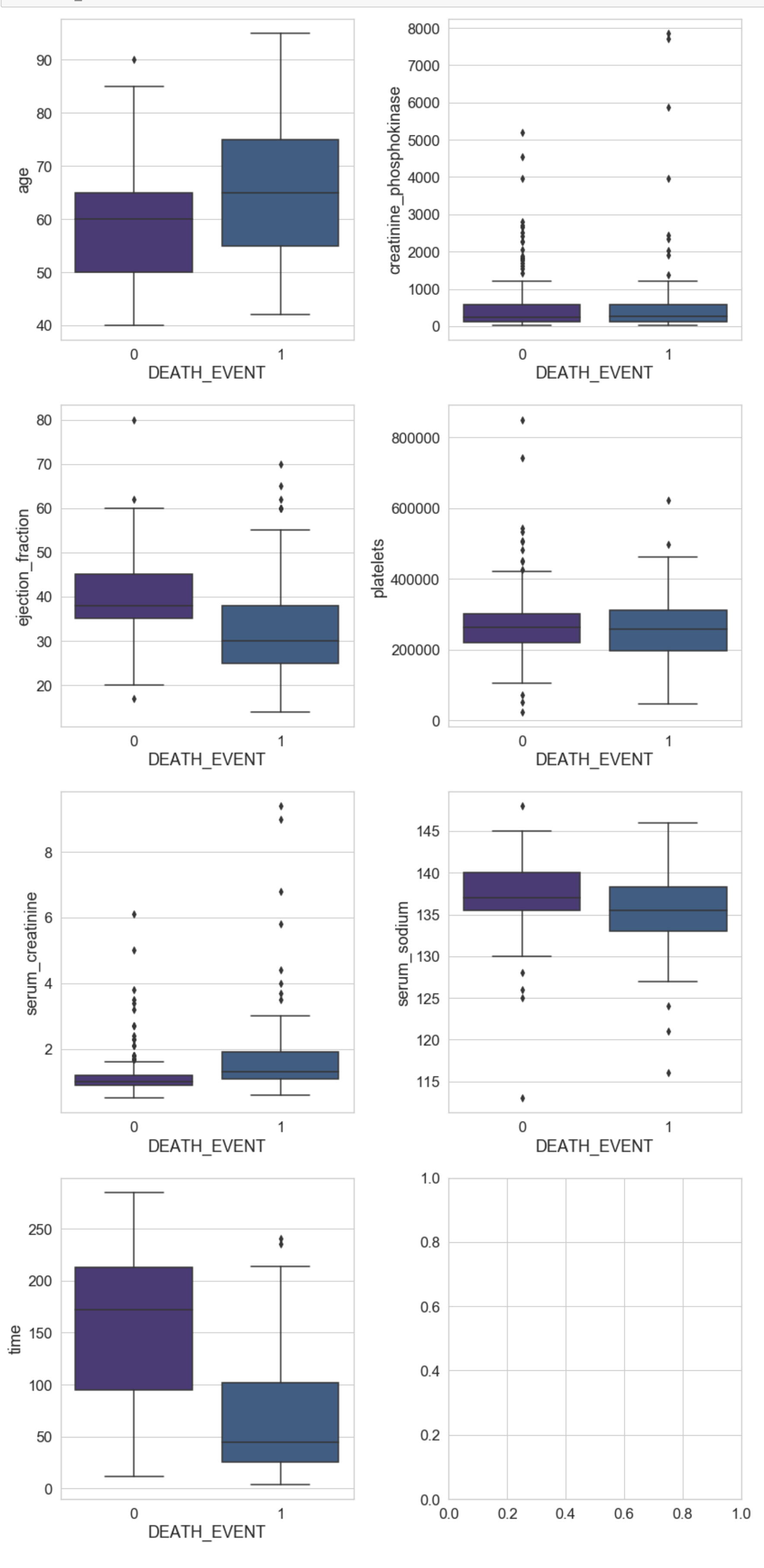

This matrix indicated variables can not highly correlated. This is good as it indicates absence of multicollinearity.

Analysis with target variable

Our objective is to predict Death Event. Analysis of the features with target variable could give us insights into the effect they could have on our predictions.

```
# Boxplots for numerical variables by risk_level
par(mfrow = c(2, 3))
boxplot(age ~ risk_level, main = "age")
boxplot(systolic_blood_pressure ~ risk_level, main = "systolic_blood_pressure")
boxplot(diastolic_blood_pressure ~ risk_level, main = "diastolic_blood_pressure")
boxplot(blood_glucose ~ risk_level, main = "blood_glucose")
boxplot(body_temperature ~ risk_level, main = "body_temperature")
boxplot(heart_rate ~ risk_level, main = "heart_rate")
```

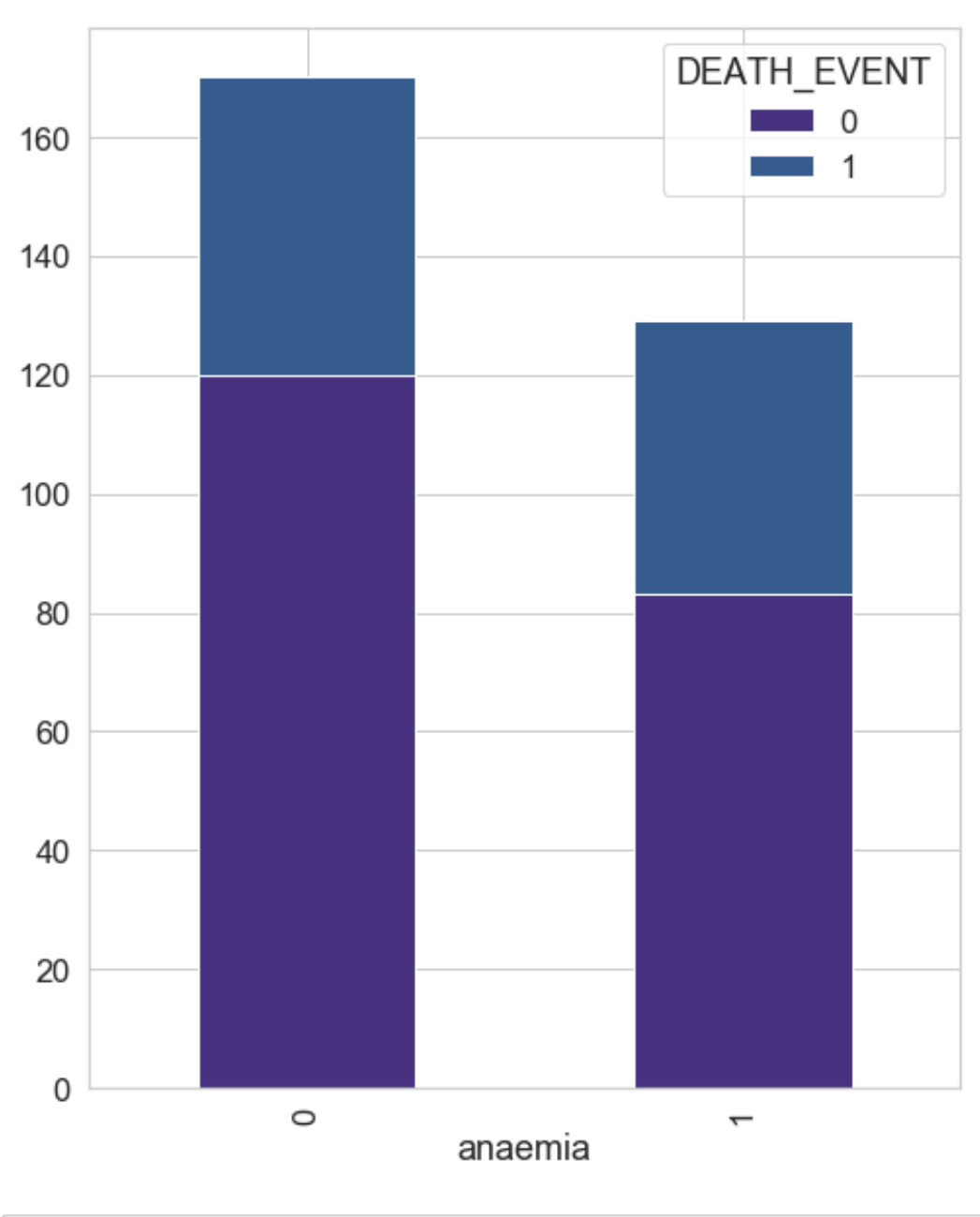
```
In [169]: fig, ax = plt.subplots(4, 2, figsize=(12, 24))
          for variable, subplot in zip(numerical, ax.flatten()):
              sns.boxplot(x = hf_data['DEATH_EVENT'] , y = hf_data_EDA[variable], ax=subplot)
          fig.tight_layout()
```



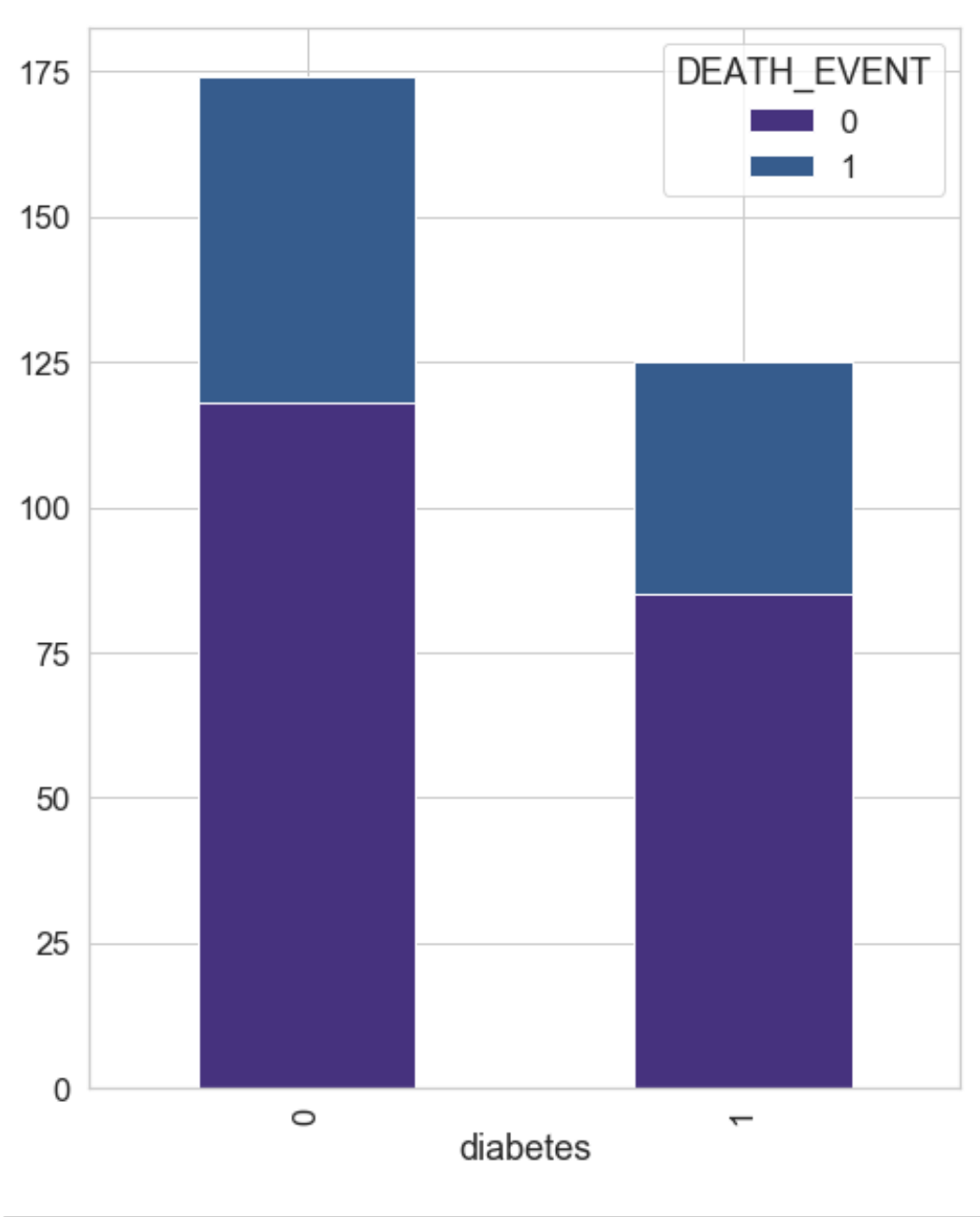
Categorical Features

```
In [168]: hf_data_EDA[categorical].columns

Out[168]: Index(['anaemia', 'diabetes', 'high_blood_pressure', 'sex', 'smoking',
                 'DEATH_EVENT'],
                dtype='object')
```

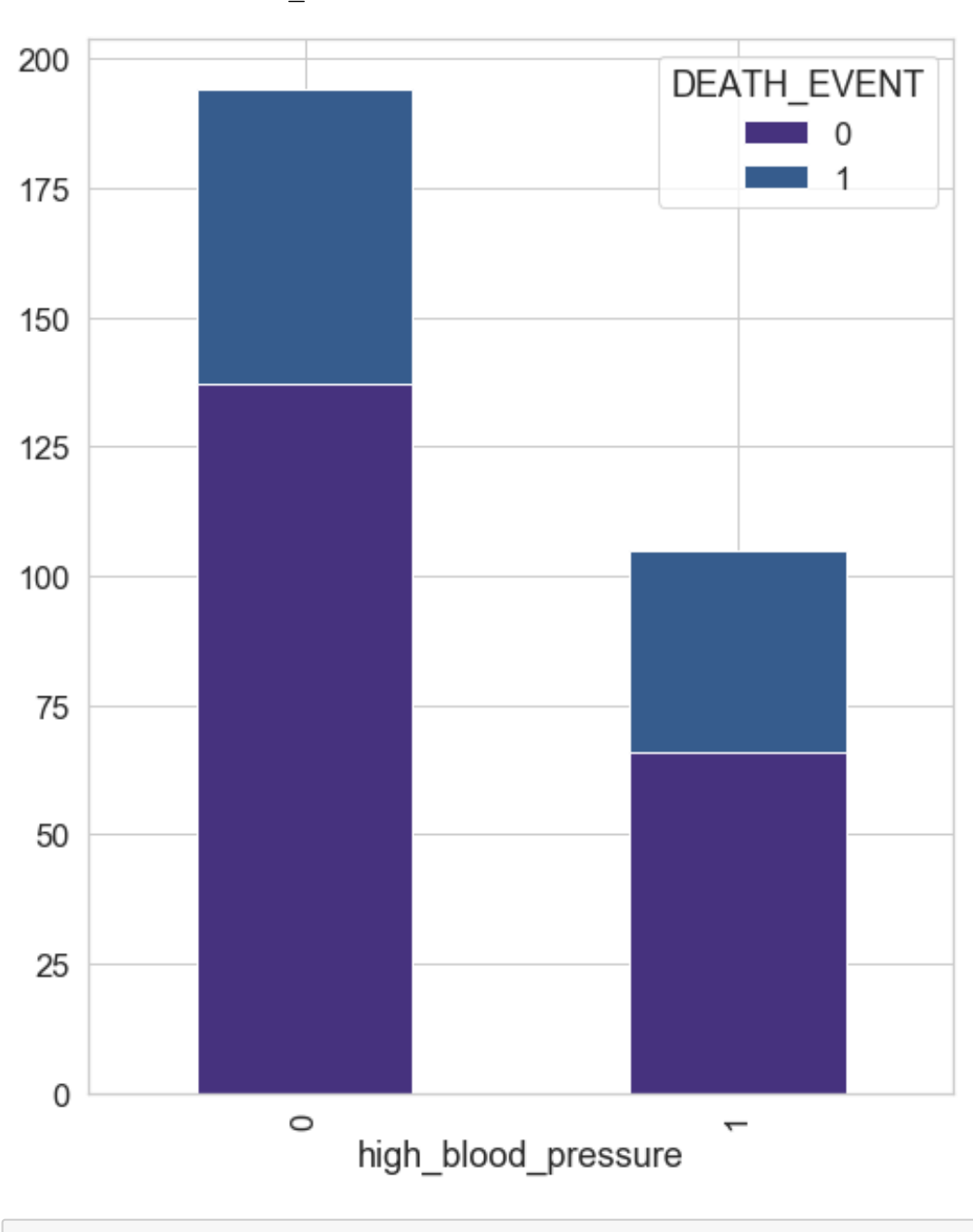
```
In [172]: counts = hf_data.groupby(['anaemia', 'DEATH_EVENT']).anaemia.count().unstack()
          counts.plot(kind='bar', stacked=True)

Out[172]: <matplotlib.axes._subplots.AxesSubplot at 0x1ac8585dc18>
```



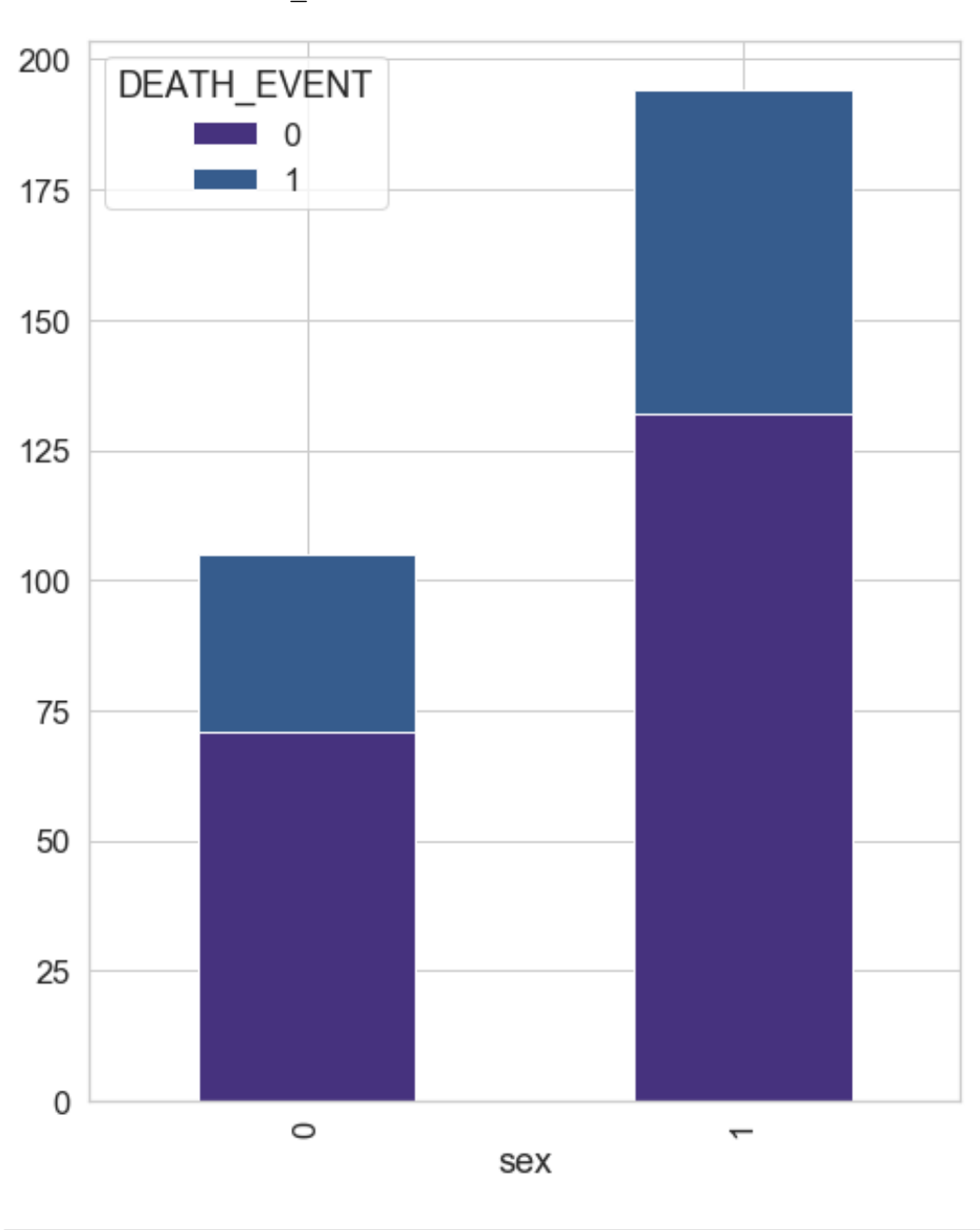
```
In [171]: counts = hf_data.groupby(['diabetes', 'DEATH_EVENT']).diabetes.count().unstack()
          counts.plot(kind='bar', stacked=True)

Out[171]: <matplotlib.axes._subplots.AxesSubplot at 0x1ac8510ac88>
```



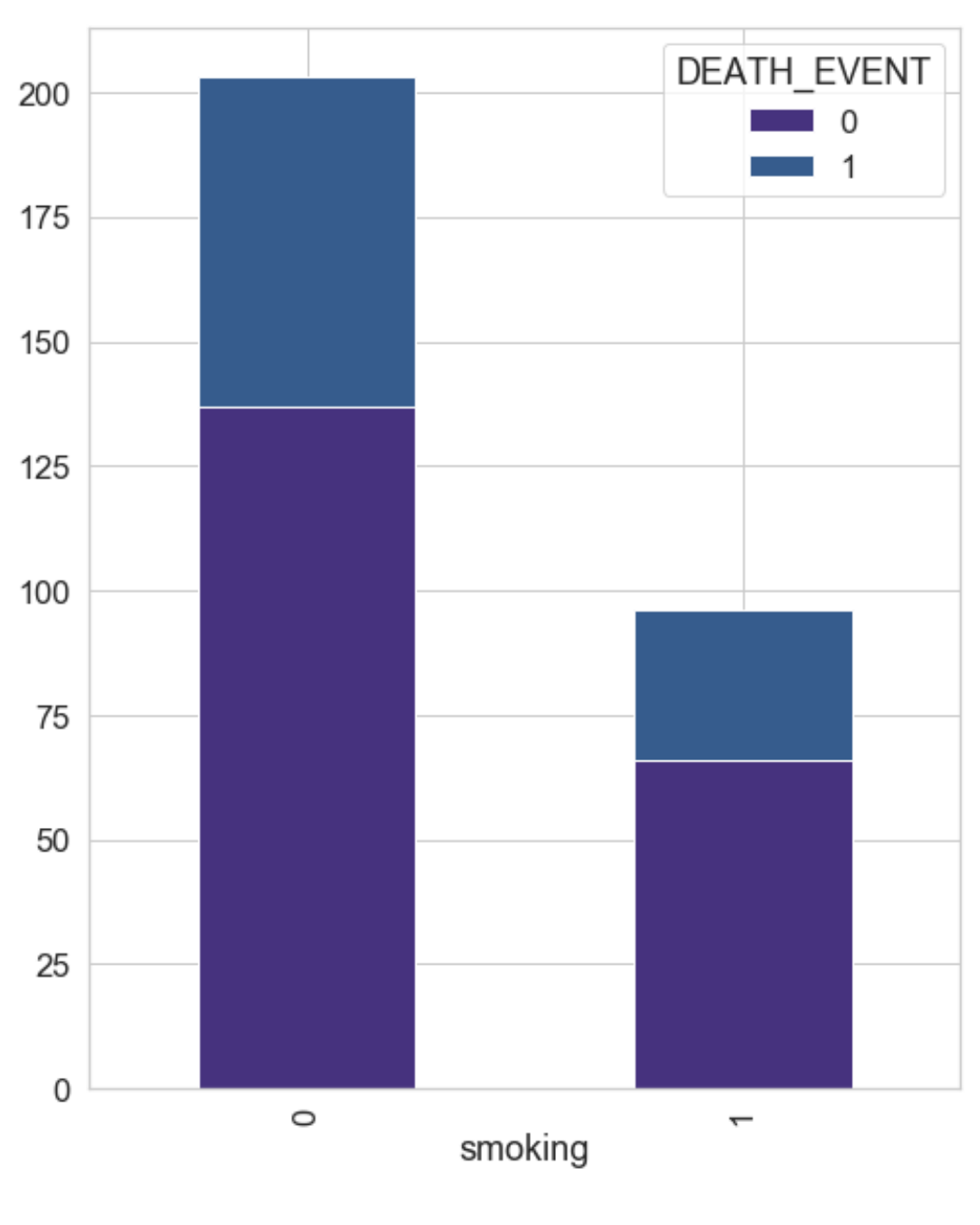
```
In [173]: counts = hf_data.groupby(['high_blood_pressure', 'DEATH_EVENT']).high_blood_pressure.count().unstack()
          counts.plot(kind='bar', stacked=True)

Out[173]: <matplotlib.axes._subplots.AxesSubplot at 0x1ac8bbf8ba8>
```



```
In [174]: counts = hf_data.groupby(['sex', 'DEATH_EVENT']).sex.count().unstack()
          counts.plot(kind='bar', stacked=True)

Out[174]: <matplotlib.axes._subplots.AxesSubplot at 0x1ac8bfef668>
```



```
In [175]: counts = hf_data.groupby(['smoking', 'DEATH_EVENT']).smoking.count().unstack()
          counts.plot(kind='bar', stacked=True)

Out[175]: <matplotlib.axes._subplots.AxesSubplot at 0x1ac9a5c5828>
```



2.1.3 Multivariate Analysis

```
In [256]: num_mva = hf_data[['age', 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_creatinine',
                             'serum_sodium','time', 'DEATH_EVENT']]
```

```
In [257]: sns.pairplot(num_mva,palette = 'bright',hue='DEATH_EVENT')

          C:\Users\Dell\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:487: RuntimeWarning:

          invalid value encountered in true_divide

          C:\Users\Dell\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetools.py:34: RuntimeWarning:

          invalid value encountered in double_scalars

          C:\Users\Dell\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:487: RuntimeWarning:

          invalid value encountered in true_divide

          C:\Users\Dell\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetools.py:34: RuntimeWarning:

          invalid value encountered in double_scalars

Out[257]: <seaborn.axisgrid.PairGrid at 0x1ac96752908>
```

2.2 Hypotheis testing

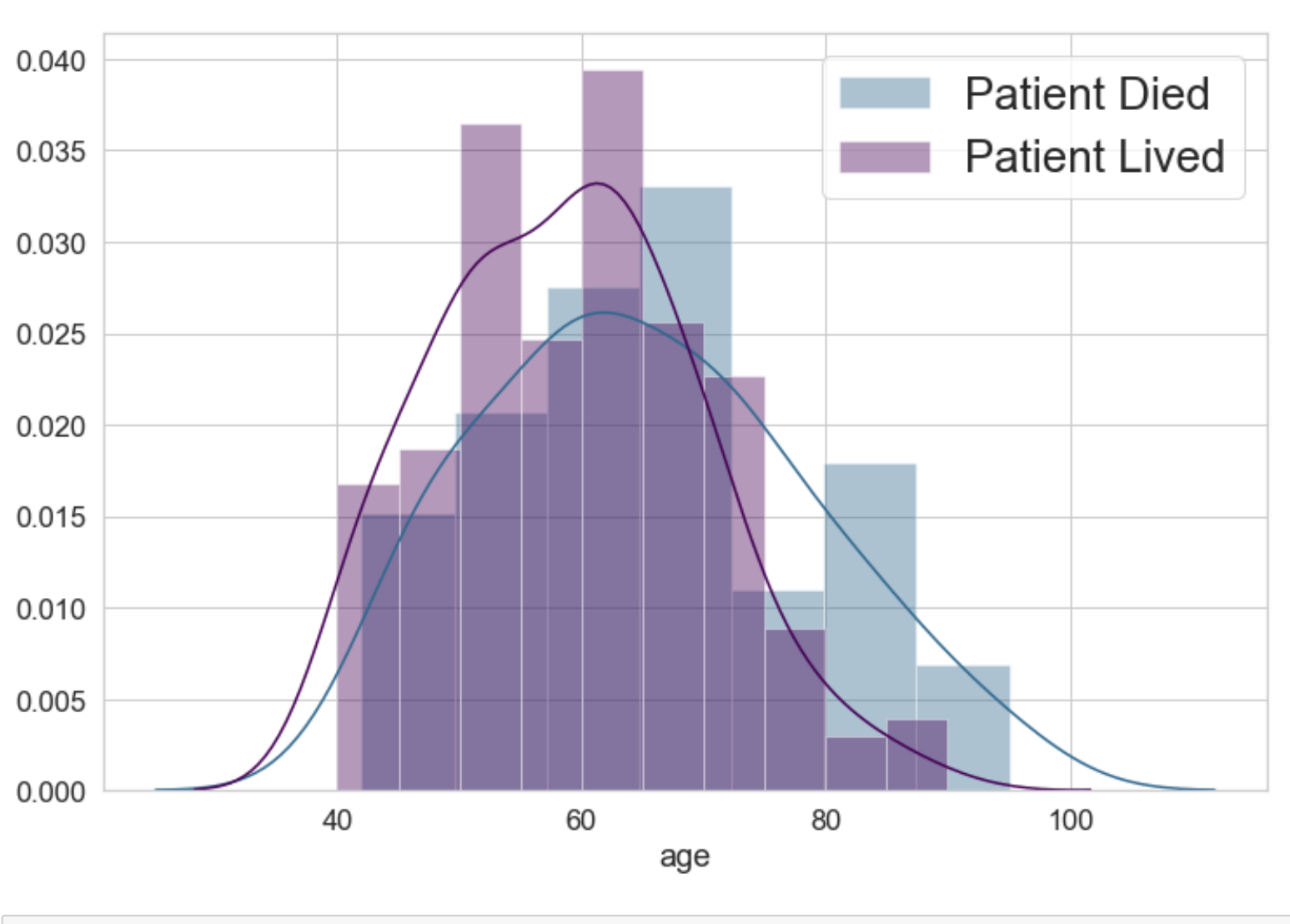
2.2.1 Numerical Variables

For comparing the distrbituion of numeric features across target variable we have used the Mann-Whitney test. It is the non-parametric equivalent of the independent samples t-test. This test has no underlying assumption regarding the normal distribution of data unlinke t-test[8].

H0: There is no statiscally significant difference between the samples
H1: There is statiscally significant difference between the samples

Age

```
In [272]: plt.figure(figsize=(12,8))
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 1]['age'],color = '#31688e', kde=True, norm_hist = False,label = 'Patient Died')
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 0]['age'],color = '#440154',kde=True, norm_hist = False,label = 'Patient Lived')
          plt.legend(fontsize = "x-large")

Out[272]: <matplotlib.legend.Legend at 0x1ac863dd748>
```

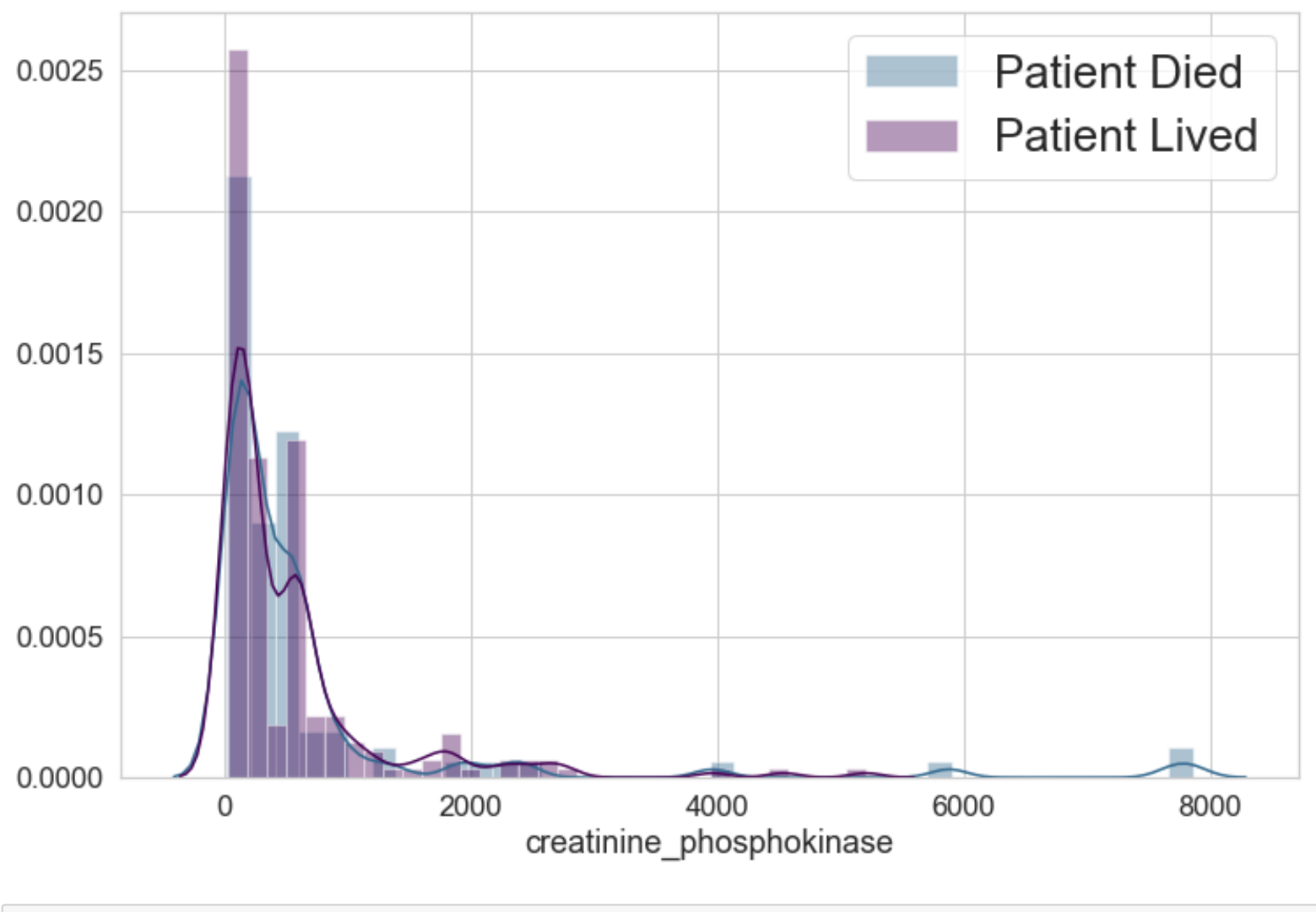


```
In [296]: from numpy.random import seed
          from numpy.random import randn
          from scipy.stats import mannwhitneyu

          patient_died = hf_data[hf_data['DEATH_EVENT'] == 1]['age']
          patient_lived = hf_data[hf_data['DEATH_EVENT'] == 0]['age']

          print("patient died: ",round(patient_died.mean(),2))
          print("patient lived: ",round(patient_lived.mean(),2))

          # comparing samples
          stat, p = mannwhitneyu(patient_died, patient_lived)
          print('Statistics=%.3f, p=%.3f' % (stat, p))
          # interpreting
          alpha = 0.05
          if p > alpha:
              print('Same distribution (fail to reject H0)')
          else:
              print('Different distribution (reject H0)')

          patient died:  65.21
          patient lived:  58.76
          Statistics=7124.500, p=0.000
          Different distribution (reject H0)
```

The patients who died are relatively older.

Creatinine Phosphokinase

```
In [277]: plt.figure(figsize=(12,8))
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 1]['creatinine_phosphokinase'],color = '#31688e', kde=True, norm_hist = False,label = 'Patient Died')
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 0]['creatinine_phosphokinase'],color = '#440154',kde=True, norm_hist = False,label = 'Patient Lived')
          plt.legend(fontsize = "x-large")

Out[277]: <matplotlib.legend.Legend at 0x1ac934a6ac8>
```

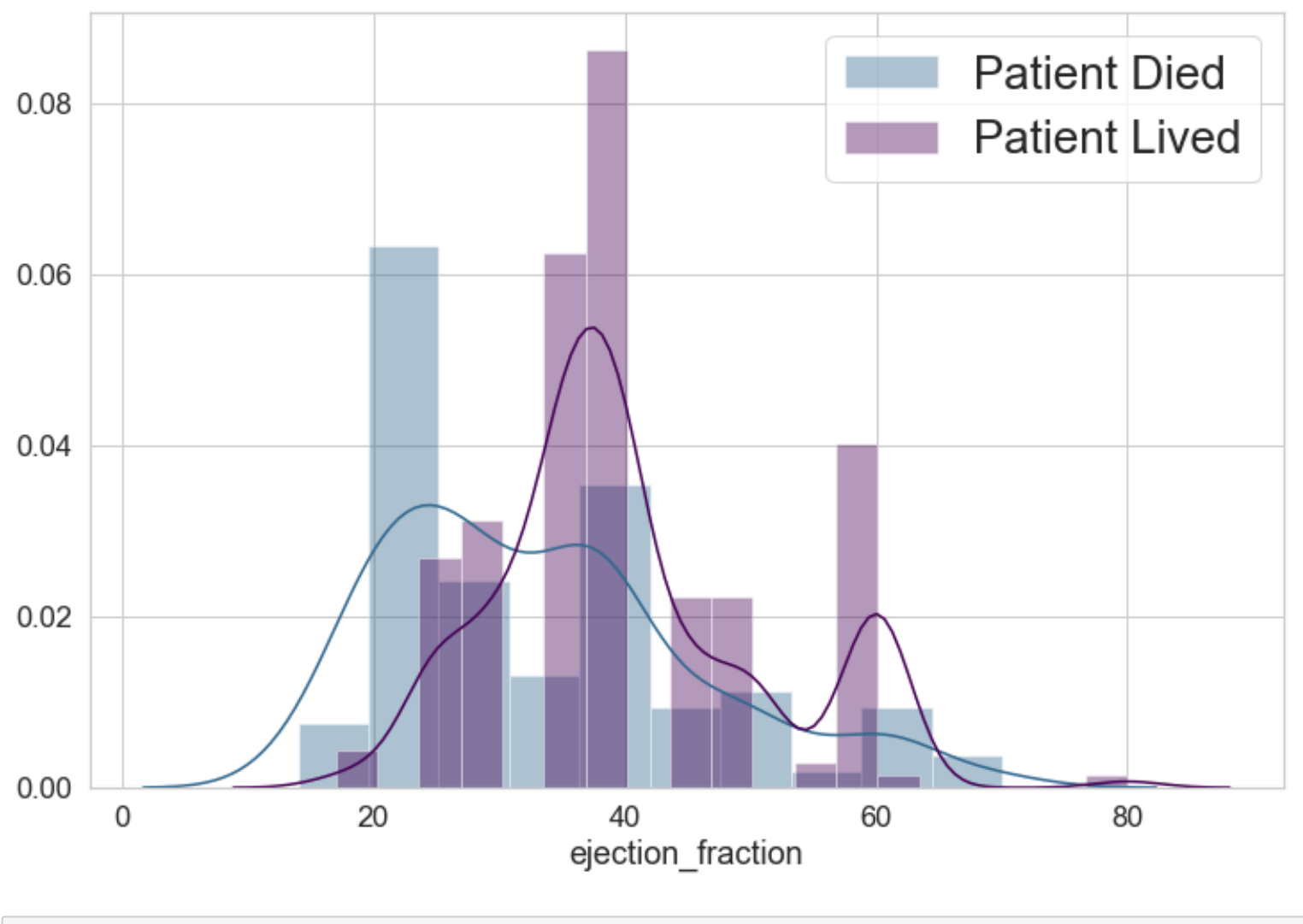


```
In [297]: patient_died = hf_data[hf_data['DEATH_EVENT'] == 1]['creatinine_phosphokinase']
          patient_lived = hf_data[hf_data['DEATH_EVENT'] == 0]['creatinine_phosphokinase']

          print("patient died: ",round(patient_died.mean(),2))
          print("patient lived: ",round(patient_lived.mean(),2))

          # comparing samples
          stat, p = mannwhitneyu(patient_died, patient_lived)
          print('Statistics=%.3f, p=%.3f' % (stat, p))
          # interpreting
          alpha = 0.05
          if p > alpha:
              print('Same distribution (fail to reject H0)')
          else:
              print('Different distribution (reject H0)')

          patient died:  670.2
          patient lived:  540.05
          Statistics=9460.000, p=0.342
          Same distribution (fail to reject H0)
```

The level of CPK enzyme is not higher among patients who died .

Ejection Fraction

```
In [280]: plt.figure(figsize=(12,8))
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 1]['ejection_fraction'],color = '#31688e', kde=True, norm_hist = False,label = 'Patient Died')
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 0]['ejection_fraction'],color = '#440154',kde=True, norm_hist = False,label = 'Patient Lived')
          plt.legend(fontsize = "x-large")

Out[280]: <matplotlib.legend.Legend at 0x1ac9354fe10>
```

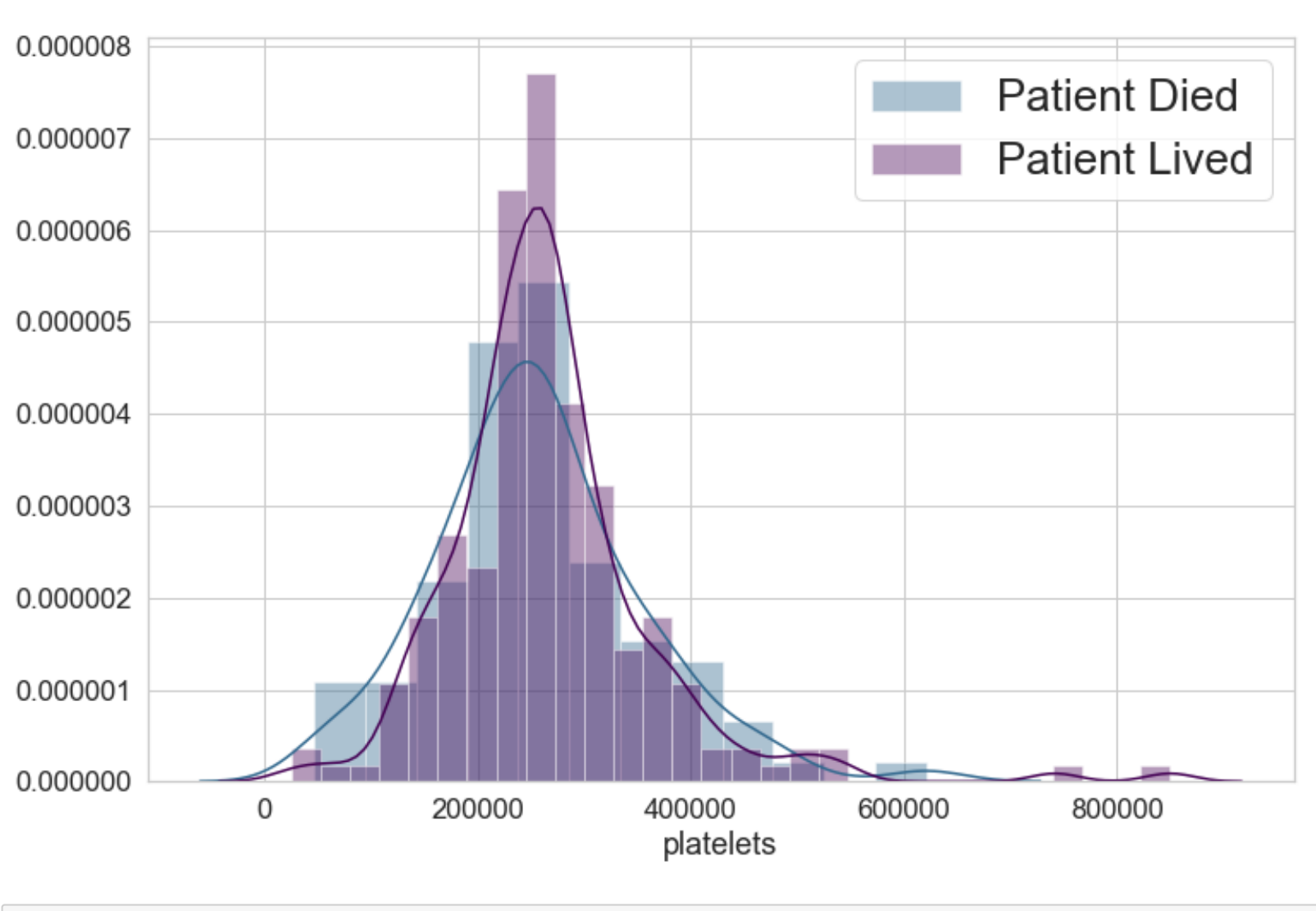


```
In [299]: patient_died = hf_data[hf_data['DEATH_EVENT'] == 1]['ejection_fraction']
          patient_lived = hf_data[hf_data['DEATH_EVENT'] == 0]['ejection_fraction']

          print("patient died: ",round(patient_died.mean(),2))
          print("patient lived: ",round(patient_lived.mean(),2))

          # comparing samples
          stat, p = mannwhitneyu(patient_died, patient_lived)
          print('Statistics=%.3f, p=%.3f' % (stat, p))
          # interpreting
          alpha = 0.05
          if p > alpha:
              print('Same distribution (fail to reject H0)')
          else:
              print('Different distribution (reject H0)')

          patient died:  33.47
          patient lived:  40.27
          Statistics=6311.500, p=0.000
          Different distribution (reject H0)
```

The ejection fraction was lower among the patients who died.

Platelets

```
In [301]: plt.figure(figsize=(12,8))
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 1]['platelets'],color = '#31688e', kde=True, norm_hist = False,label = 'Patient Died')
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 0]['platelets'],color = '#440154',kde=True, norm_hist = False,label = 'Patient Lived')
          plt.legend(fontsize = "x-large")

Out[301]: <matplotlib.legend.Legend at 0x1ac954fce80>
```

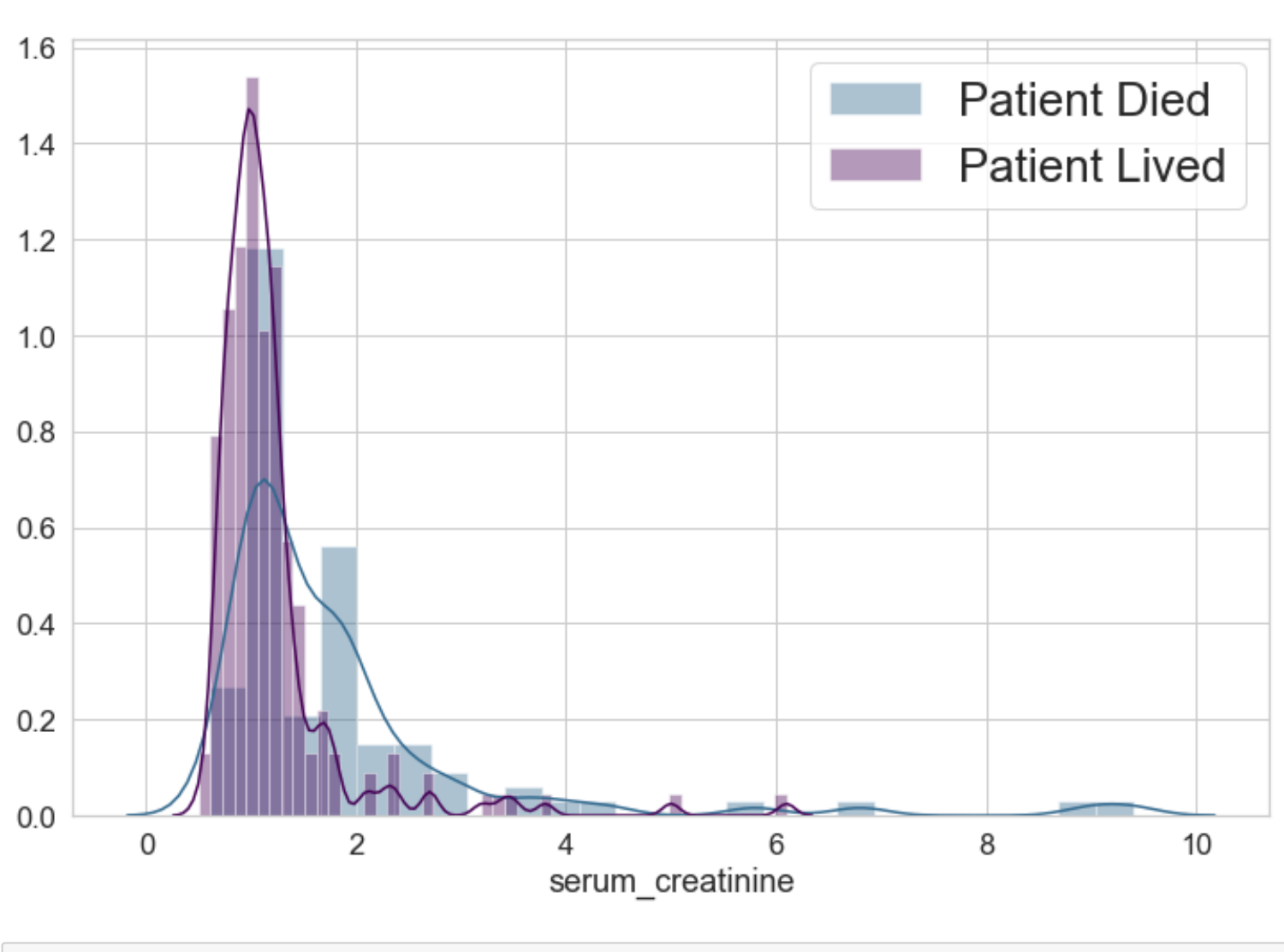


```
In [305]: patient_died = hf_data[hf_data['DEATH_EVENT'] == 1]['platelets']
          patient_lived = hf_data[hf_data['DEATH_EVENT'] == 0]['platelets']

          print("patient died: ",round(patient_died.mean(),2))
          print("patient lived: ",round(patient_lived.mean(),2))

          # comparing samples
          stat, p = mannwhitneyu(patient_died, patient_lived)
          print('Statistics=%.3f, p=%.3f' % (stat, p))
          # interpreting
          alpha = 0.05
          if p > alpha:
              print('Same distribution (fail to reject H0)')
          else:
              print('Different distribution (reject H0)')

          patient died:  256381.04
          patient lived:  266657.49
          Statistics=9187.500, p=0.213
          Same distribution (fail to reject H0)
```

The difference between the average level of plateletes is isignificant.

Serum Creatinine

```
In [303]: plt.figure(figsize=(12,8))
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 1]['serum_creatinine'],color = '#31688e', kde=True, norm_hist = False,label = 'Patient Died')
          sns.distplot(hf_data[hf_data['DEATH_EVENT'] == 0]['serum_creatinine'],color = '#440154',kde=True, norm_hist = False,label = 'Patient Lived')
          plt.legend(fontsize = "x-large")

Out[303]: <matplotlib.legend.Legend at 0x1ac94c6ac18>
```



```
In [307]: patient_died = hf_data[hf_data['DEATH_EVENT'] == 1]['serum_creatinine']
          patient_lived = hf_data[hf_data['DEATH_EVENT'] == 0]['serum_creatinine']

          print("patient died: ",round(patient_died.mean(),2))
          print("patient lived: ",round(patient_lived.mean(),2))

          # comparing samples
          stat, p = mannwhitneyu(patient_died, patient_lived)
          print('Statistics=%.3f, p=%.3f' % (stat, p))
          # interpreting
          alpha = 0.05
          if p > alpha:
              print('Same distribution (fail to reject H0)')
          else:
              print('Different distribution (reject H0)')

          patient died:  1.84
          patient lived:  1.18
          Statistics=5298.000, p=0.000
          Different distribution (reject H0)
```

The levels of serum creatinine was higher on average among the patients who died.

Serum Sodium

```
data_plot.dropna(inplace=True)
sns.boxplot(x="heart_disease", y="avg_glucose_level", data=data_plot, palette=["#9999ff", "ffb366"], linewidth=1.5, fliersize=3)
plt.xlabel("Heart Disease", fontsize=12)
plt.ylabel("Average Glucose Level", fontsize=12)
plt.xticks(ticks=[0,1], labels=["Patient Died", "Patient Lived"])
plt.title("Average Glucose Level Vs Heart Disease", fontsize=14, pad=20)
plt.show()
```

Average Glucose Level Vs Heart Disease


```
patient_died = data[data['heart_disease']==1]['avg_glucose_level']
patient_lived = data[data['heart_disease']==0]['avg_glucose_level']
print("Patient Died: {:f}".format(patient_died.mean()))
print("Patient Lived: {:f}".format(patient_lived.mean()))
# Computing statistic
numerator = patient_died.mean() - patient_lived.mean()
denominator = np.sqrt(patient_died.var()/len(patient_died) + patient_lived.var()/len(patient_lived))
print("{:f}".format(numerator/denominator))
# p-value
df = len(patient_died)+len(patient_lived)-2
print("{:f}".format(stats.t.cdf(-(numerator/denominator), df)*2))
```

```
Patient Died: 132.544739
Patient Lived: 104.795889
7.954559
0.000000
```

The mean of severe cerebrovascular lesion or average among the patients who died.

2.2 Categorical Variables

Chi-square goodness-of-fit test is a non-parametric test used for comparing the observed distribution of data with the expected distribution of data. The chi-square test tells us how much difference exists between the observed and expected data. This statistical distribution and a theoretical distribution.

Chi Square is a difference between observed and expected frequencies.

We have a difference between observed and expected frequencies.

Gender

```
# Cross-tabulation (contingency table)
ct = pd.crosstab(data['heart_disease'], data['gender'])
print(ct)
from scipy.stats import chi2_contingency
print("Test statistic, p-value, degrees of freedom, expected frequencies: \n",chi2_contingency(ct))
# p-value
print("p-value: {:f}".format(chi2_contingency(ct)[1]))
# Test statistic
print("Test statistic: {:f}".format(chi2_contingency(ct)[0]))
# degrees of freedom
print("Degrees of freedom: {:f}".format(chi2_contingency(ct)[2]))
# expected frequencies
print("Expected frequencies: \n",chi2_contingency(ct)[3])
```

```
gender  Female  Male  Other
heart_disease
0        2897  2011      1
1         141   108      0
Test statistic, p-value, degrees of freedom, expected frequencies:
 (0.0, 1.0, 2, array([[2896.0,  2011.0,     1.0],
       [ 142.0,   107.0,     0.0]]))
p-value: 1.000000
Test statistic: 0.000000
Degrees of freedom: 2.000000
Expected frequencies:
 [[2896.0  2011.0     1.0]
 [ 142.0   107.0     0.0]]
```

The dependent variable is not dependent on the gender.

Hypertension

```
# Cross-tabulation (contingency table)
ct = pd.crosstab(data['heart_disease'], data['hypertension'])
print(ct)
from scipy.stats import chi2_contingency
print("Test statistic, p-value, degrees of freedom, expected frequencies: \n",chi2_contingency(ct))
# p-value
print("p-value: {:f}".format(chi2_contingency(ct)[1]))
# Test statistic
print("Test statistic: {:f}".format(chi2_contingency(ct)[0]))
# degrees of freedom
print("Degrees of freedom: {:f}".format(chi2_contingency(ct)[2]))
# expected frequencies
print("Expected frequencies: \n",chi2_contingency(ct)[3])
```

```
hypertension   0    1
heart_disease
0           4612  297
1            232   17
Test statistic, p-value, degrees of freedom, expected frequencies:
 (0.0, 1.0, 1, array([[4612.0,  297.0],
       [ 232.0,   17.0]]))
p-value: 1.000000
Test statistic: 0.000000
Degrees of freedom: 1.000000
Expected frequencies:
 [[4612.  297.]
 [ 232.   17.]]
```

The dependent variable is not dependent on the hypertension.

Heart Disease

```
# Cross-tabulation (contingency table)
ct = pd.crosstab(data['heart_disease'], data['heart_disease'])
print(ct)
from scipy.stats import chi2_contingency
print("Test statistic, p-value, degrees of freedom, expected frequencies: \n",chi2_contingency(ct))
# p-value
print("p-value: {:f}".format(chi2_contingency(ct)[1]))
# Test statistic
print("Test statistic: {:f}".format(chi2_contingency(ct)[0]))
# degrees of freedom
print("Degrees of freedom: {:f}".format(chi2_contingency(ct)[2]))
# expected frequencies
print("Expected frequencies: \n",chi2_contingency(ct)[3])
```

```
heart_disease  0    1
heart_disease
0           4909    0
1              0  249
Test statistic, p-value, degrees of freedom, expected frequencies:
 (4909.000000000001, 0.0, 1, array([[4909.,    0.],
       [   0.,  249.]]))
p-value: 0.000000
Test statistic: 4909.000000
Degrees of freedom: 1.000000
Expected frequencies:
 [[4909.    0.]
 [   0.  249.]]
```

The dependent variable is not dependent on the heart_disease.

Ever_married

```
# Cross-tabulation (contingency table)
ct = pd.crosstab(data['heart_disease'], data['ever_married'])
print(ct)
from scipy.stats import chi2_contingency
print("Test statistic, p-value, degrees of freedom, expected frequencies: \n",chi2_contingency(ct))
# p-value
print("p-value: {:f}".format(chi2_contingency(ct)[1]))
# Test statistic
print("Test statistic: {:f}".format(chi2_contingency(ct)[0]))
# degrees of freedom
print("Degrees of freedom: {:f}".format(chi2_contingency(ct)[2]))
# expected frequencies
print("Expected frequencies: \n",chi2_contingency(ct)[3])
```

```
ever_married   No  Yes
heart_disease
0           1728 3181
1             89  160
Test statistic, p-value, degrees of freedom, expected frequencies:
 (0.0, 1.0, 1, array([[1728.0, 3181.0],
       [  89.0,  160.0]]))
p-value: 1.000000
Test statistic: 0.000000
Degrees of freedom: 1.000000
Expected frequencies:
 [[1728. 3181.]
 [  89.  160.]]
```

The dependent variable is not dependent on the ever_married.

Work_type

```
# Cross-tabulation (contingency table)
ct = pd.crosstab(data['heart_disease'], data['work_type'])
print(ct)
from scipy.stats import chi2_contingency
print("Test statistic, p-value, degrees of freedom, expected frequencies: \n",chi2_contingency(ct))
# p-value
print("p-value: {:f}".format(chi2_contingency(ct)[1]))
# Test statistic
print("Test statistic: {:f}".format(chi2_contingency(ct)[0]))
# degrees of freedom
print("Degrees of freedom: {:f}".format(chi2_contingency(ct)[2]))
# expected frequencies
print("Expected frequencies: \n",chi2_contingency(ct)[3])
```

```
work_type  Govt_job  Never_work  Private  Self-employed  children
heart_disease
0            1126          22     2925            649      1068
1              78           2      132             37         0
Test statistic, p-value, degrees of freedom, expected frequencies:
 (0.0, 1.0, 4, array([[1126.,   22., 2925.,  649., 1068.],
       [  78.,    2.,  132.,   37.,    0.]]))
p-value: 1.000000
Test statistic: 0.000000
Degrees of freedom: 4.000000
Expected frequencies:
 [[1126.   22. 2925.  649. 1068.]
 [  78.    2.  132.   37.    0.]]
```

The dependent variable is not dependent on the work_type.

Residence_type

```
# Cross-tabulation (contingency table)
ct = pd.crosstab(data['heart_disease'], data['Residence_type'])
print(ct)
from scipy.stats import chi2_contingency
print("Test statistic, p-value, degrees of freedom, expected frequencies: \n",chi2_contingency(ct))
# p-value
print("p-value: {:f}".format(chi2_contingency(ct)[1]))
# Test statistic
print("Test statistic: {:f}".format(chi2_contingency(ct)[0]))
# degrees of freedom
print("Degrees of freedom: {:f}".format(chi2_contingency(ct)[2]))
# expected frequencies
print("Expected frequencies: \n",chi2_contingency(ct)[3])
```

```
Residence_type   Rural  Urban
heart_disease
0             2419   2490
1              126    123
Test statistic, p-value, degrees of freedom, expected frequencies:
 (0.0, 1.0, 1, array([[2419.,  2490.],
       [ 126.,   123.]]))
p-value: 1.000000
Test statistic: 0.000000
Degrees of freedom: 1.000000
Expected frequencies:
 [[2419.  2490.]
 [ 126.   123.]]
```

The dependent variable is not dependent on the Residence_type.

Smoking

```
# Cross-tabulation (contingency table)
ct = pd.crosstab(data['heart_disease'], data['smoking_status'])
print(ct)
from scipy.stats import chi2_contingency
print("Test statistic, p-value, degrees of freedom, expected frequencies: \n",chi2_contingency(ct))
# p-value
print("p-value: {:f}".format(chi2_contingency(ct)[1]))
# Test statistic
print("Test statistic: {:f}".format(chi2_contingency(ct)[0]))
# degrees of freedom
print("Degrees of freedom: {:f}".format(chi2_contingency(ct)[2]))
# expected frequencies
print("Expected frequencies: \n",chi2_contingency(ct)[3])
```

```
smoking_status  formerly smoked  never smoked  smokes  Unknown
heart_disease
0                           884          1892     747     1386
1                            70           110      42       27
Test statistic, p-value, degrees of freedom, expected frequencies:
 (0.0, 1.0, 3, array([[884.,  1892.,   747.,  1386.],
       [ 70.,   110.,    42.,    27.]]))
p-value: 1.000000
Test statistic: 0.000000
Degrees of freedom: 3.000000
Expected frequencies:
 [[884.  1892.   747.  1386.]
 [ 70.   110.    42.    27.]]
```

The dependent variable is not dependent on the smoking_status.

3.Modeling & Evaluation

The aim of our model would be to predict the event of patient's death. Since the outcome is categorical in nature, classification algorithms would be used.

Separating Dependent and Independent variables

```
# Dividing dependent and independent variables into train and test datasets
# Dependent variable
X = data[['age', 'hypertension', 'heart_disease', 'ever_married', 'avg_glucose_level', 'bmi', 'gender_Female', 'gender_Male', 'gender_Other', 'work_type_Govt_job', 'work_type_Never_work', 'work_type_Private', 'work_type_Self-employed', 'work_type_children', 'Residence_type_Rural', 'Residence_type_Urban', 'smoking_status_formerly smoked', 'smoking_status_never smoked', 'smoking_status_smokes', 'smoking_status_Unknown']]
# Independent variable
y = data['stroke']
```

Train Test Split

```
# Splitting the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,random_state = 123)
```

Logistic Regression

```
# Logistic Regression using statsmodels package from scipy.The statsmodels package includes well-established summaries for the results
# Importing libraries
import statsmodels.api as sm
# Fitting the model
# Probabilities of outcomes can be estimated
# Robust Error correction
# Coefficients are interpreted as the distributions
logit = sm.Logit(y_train, X_train)
result = logit.fit(maxiter=100, method='bfgs', disp=True)
print(result.summary())
# Coefficients
print("Coefficients: \n",result.params)
# p-values
print("p-values: \n",result.pvalues)
# Odds ratios
print("Odds ratios: \n",np.exp(result.params))
# Confidence intervals
print("Confidence intervals: \n",result.conf_int())
```

```
Optimization terminated successfully.
         Current function value: 0.141000
         Gradients have small values.
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.141000
         Iterations: 100
         Function evaluations: 102
         Gradient evaluations: 102
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 3832
Model:                          Logit   Df Residuals:                     3812
Method:                           MLE   Df Model:                           19
Date:                Sun, 28 Feb 2021   Pseudo R-squ.:                  0.2000
Time:                        17:46:47   Log-Likelihood:                -540.16
converged:                       False   LL-Null:                       -675.00
Covariance Type:                    u   LLR p-value:                     1.000
=========================================================================================
                            coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------
age                       0.0700      0.000    147.000      0.000       0.069       0.071
hypertension              0.4900      0.000    980.000      0.000       0.489       0.491
heart_disease             0.5000      0.000   1000.000      0.000       0.499       0.501
ever_married              0.0000      0.000      0.000      1.000      -0.000       0.000
avg_glucose_level         0.0050      0.000     10.000      0.000       0.004       0.006
bmi                       0.0100      0.000     20.000      0.000       0.009       0.011
gender_Female             0.0000      0.000      0.000      1.000      -0.000       0.000
gender_Male               0.0000      0.000      0.000      1.000      -0.000       0.000
gender_Other              0.0000      0.000      0.000      1.000      -0.000       0.000
work_type_Govt_job        0.0000      0.000      0.000      1.000      -0.000       0.000
work_type_Never_work      0.0000      0.000      0.000      1.000      -0.000       0.000
work_type_Private         0.0000      0.000      0.000      1.000      -0.000       0.000
work_type_Self-employed   0.0000      0.000      0.000      1.000      -0.000       0.000
work_type_children        0.0000      0.000      0.000      1.000      -0.000       0.000
Residence_type_Rural      0.0000      0.000      0.000      1.000      -0.000       0.000
Residence_type_Urban      0.0000      0.000      0.000      1.000      -0.000       0.000
smoking_status_formerly smoked   0.0000      0.000      0.000      1.000      -0.000       0.000
smoking_status_never smoked      0.0000      0.000      0.000      1.000      -0.000       0.000
smoking_status_smokes            0.0000      0.000      0.000      1.000      -0.000       0.000
smoking_status_Unknown           0.0000      0.000      0.000      1.000      -0.000       0.000
=========================================================================================
Coefficients:
 age                       0.069900
hypertension              0.490000
heart_disease             0.500000
ever_married              0.000000
avg_glucose_level         0.005000
bmi                       0.010000
gender_Female             0.000000
gender_Male               0.000000
gender_Other              0.000000
work_type_Govt_job        0.000000
work_type_Never_work      0.000000
work_type_Private         0.000000
work_type_Self-employed   0.000000
work_type_children        0.000000
Residence_type_Rural      0.000000
Residence_type_Urban      0.000000
smoking_status_formerly smoked   0.000000
smoking_status_never smoked      0.000000
smoking_status_smokes            0.000000
smoking_status_Unknown           0.000000
dtype: float64
p-values:
 age                       0.000000
hypertension              0.000000
heart_disease             0.000000
ever_married              1.000000
avg_glucose_level         0.000000
bmi                       0.000000
gender_Female             1.000000
gender_Male               1.000000
gender_Other              1.000000
work_type_Govt_job        1.000000
work_type_Never_work      1.000000
work_type_Private         1.000000
work_type_Self-employed   1.000000
work_type_children        1.000000
Residence_type_Rural      1.000000
Residence_type_Urban      1.000000
smoking_status_formerly smoked   1.000000
smoking_status_never smoked      1.000000
smoking_status_smokes            1.000000
smoking_status_Unknown           1.000000
dtype: float64
Odds ratios:
 age                       1.072400
hypertension              1.632000
heart_disease             1.649000
ever_married              1.000000
avg_glucose_level         1.005000
bmi                       1.010000
gender_Female             1.000000
gender_Male               1.000000
gender_Other              1.000000
work_type_Govt_job        1.000000
work_type_Never_work      1.000000
work_type_Private         1.000000
work_type_Self-employed   1.000000
work_type_children        1.000000
Residence_type_Rural      1.000000
Residence_type_Urban      1.000000
smoking_status_formerly smoked   1.000000
smoking_status_never smoked      1.000000
smoking_status_smokes            1.000000
smoking_status_Unknown           1.000000
dtype: float64
Confidence intervals:
                           0.025     0.975
age                       0.069     0.071
hypertension              0.489     0.491
heart_disease             0.499     0.501
ever_married             -0.000     0.000
avg_glucose_level         0.004     0.006
bmi                       0.009     0.011
gender_Female            -0.000     0.000
gender_Male              -0.000     0.000
gender_Other             -0.000     0.000
work_type_Govt_job       -0.000     0.000
work_type_Never_work     -0.000     0.000
work_type_Private        -0.000     0.000
work_type_Self-employed  -0.000     0.000
work_type_children       -0.000     0.000
Residence_type_Rural     -0.000     0.000
Residence_type_Urban     -0.000     0.000
smoking_status_formerly smoked  -0.000     0.000
smoking_status_never smoked     -0.000     0.000
smoking_status_smokes           -0.000     0.000
smoking_status_Unknown          -0.000     0.000
```

```
# Cross-validation
# Importing libraries
from sklearn.model_selection import cross_val_score
# Fitting the model
logreg = LogisticRegression(fit_intercept=True, max_iter=10000)
cv_lr = cross_val_score(logreg, X_train, y_train, cv=5)
# Printing the mean and standard deviation of the cross-validation scores
print("Mean: {:f}".format(np.mean(cv_lr)))
print("Standard Deviation: {:f}".format(np.std(cv_lr)))
```

```
Mean: 0.946000
Standard Deviation: 0.000000
```

```
# Predicting probabilities of Test data
y_pred = logreg.predict(X_test)
print("Predictions: \n", y_pred)
```

```
Predictions:
 [0 0 0 ... 0 0 0]
```

```
y_pred_df.value_counts(normalize=True)
```

```
0    0.951
1    0.049
dtype: float64
```

```
# Confusion matrix
# A table which is used to summarize the classic performance of a classification algorithm model and count
# Importing libraries
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score, auc, precision_recall_curve, average_precision_score
y_pred_df.value_counts(normalize=True)
# Creating a dataset with predicted and actual values
y_pred = logreg.predict(X_test)
y_pred_df = pd.DataFrame(y_pred, columns=['predicted'])
y_pred_df.value_counts(normalize=True)
# Confusion matrix
print("Confusion matrix: \n",confusion_matrix(y_test, y_pred))
# Classification report
print("Classification report: \n",classification_report(y_test, y_pred))
# Plot ROC curve
y_pred_prob = logreg.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
# Area under ROC curve
print("AUC: {:f}".format(roc_auc_score(y_test, y_pred_prob)))
```

```
Confusion matrix:
 [[1215    0]
 [  62    0]]
Classification report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1215
           1       0.00      0.00      0.00        62

    accuracy                           0.95      1277
   macro avg       0.47      0.50      0.49      1277
weighted avg       0.90      0.95      0.92      1277
AUC: 0.823000
```


```
# Confusion matrix plot
# Importing libraries
from sklearn.metrics import plot_confusion_matrix
# Plot confusion matrix
plot_confusion_matrix(logreg, X_test, y_test, cmap=plt.cmap.Blues, values_format = 'd')
```



```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(logreg, X_train, y_train, cv=5)
# Confusion matrix
print("Confusion matrix: \n",confusion_matrix(y_train, cv_pred))
# Classification report
print("Classification report: \n",classification_report(y_train, cv_pred))
# Plot ROC curve
cv_pred_prob = cross_val_predict(logreg, X_train, y_train, cv=5, method='predict_proba')[:,1]
fpr, tpr, thresholds = roc_curve(y_train, cv_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
# Area under ROC curve
print("AUC: {:f}".format(roc_auc_score(y_train, cv_pred_prob)))
```

```
Confusion matrix:
 [[3646    0]
 [ 186    0]]
Classification report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      3646
           1       0.00      0.00      0.00       186

    accuracy                           0.95      3832
   macro avg       0.47      0.50      0.49      3832
weighted avg       0.90      0.95      0.92      3832
AUC: 0.817000
```



```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(logreg, X_train, y_train, cv=5)
# Plot confusion matrix
plot_confusion_matrix(logreg, X_train, y_train, cmap=plt.cmap.Blues, values_format = 'd')
```



```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(logreg, X_train, y_train, cv=5)
# Plot normalized confusion matrix
plot_confusion_matrix(logreg, X_train, y_train, cmap=plt.cmap.Blues, values_format = '.2f', normalize = 'true')
```



```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(logreg, X_train, y_train, cv=5)
# Plot normalized confusion matrix
plot_confusion_matrix(logreg, X_train, y_train, cmap=plt.cmap.Blues, values_format = '.2f', normalize = 'pred')
```


3.2 Naive-Bayes Classifier

Naive Bayes classifiers are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

Gaussian Naive Bayes

1. Binomial and non-binomial variable
2. Presence binomial conditional probabilities
3. Continuous variable conditional probabilities follow normal distribution

```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(GaussianNB(), X_train, y_train, cv=5)
# Plot confusion matrix
plot_confusion_matrix(GaussianNB(), X_train, y_train, cmap=plt.cmap.Blues, values_format = 'd')
```



```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(GaussianNB(), X_train, y_train, cv=5)
# Plot normalized confusion matrix
plot_confusion_matrix(GaussianNB(), X_train, y_train, cmap=plt.cmap.Blues, values_format = '.2f', normalize = 'true')
```



```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(GaussianNB(), X_train, y_train, cv=5)
# Plot normalized confusion matrix
plot_confusion_matrix(GaussianNB(), X_train, y_train, cmap=plt.cmap.Blues, values_format = '.2f', normalize = 'pred')
```



```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(GaussianNB(), X_train, y_train, cv=5)
# Plot ROC curve
cv_pred_prob = cross_val_predict(GaussianNB(), X_train, y_train, cv=5, method='predict_proba')[:,1]
fpr, tpr, thresholds = roc_curve(y_train, cv_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Gaussian Naive Bayes')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Gaussian Naive Bayes ROC Curve')
# Area under ROC curve
print("AUC: {:f}".format(roc_auc_score(y_train, cv_pred_prob)))
```

```
AUC: 0.817000
```



```
# Cross-validation for cross predictions
# Importing libraries
from sklearn.model_selection import cross_val_predict
# Fitting the model
cv_pred = cross_val_predict(GaussianNB(), X_train, y_train, cv=5)
# Classification report
print("Classification report: \n",classification_report(y_train, cv_pred))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      3646
           1       0.13      0.13      0.13       186

    accuracy                           0.90      3832
   macro avg       0.54      0.54      0.54      3832
weighted avg       0.90      0.90      0.90      3832
```

```
In [24]: y_pred_DT=dt_model.predict(X_test)
         print("Predicted values:")
         y_pred_DT[0:5]
         print("Confusion matrix")
         confusion_DT=confusion_matrix(Y_test,y_pred_DT)
         print(confusion_DT)
         print("Classification report")
         print(classification_report(Y_test,y_pred_DT))
         sns.heatmap(confusion_DT,annot=True,cmap="Spectral")
         plt.xlabel('Predicted Class',fontsize=15)
         plt.ylabel('True Class',fontsize=15)
         plt.title('Confusion Matrix',fontsize=15)
         ax = plt.axis([-0.5, 1.5, 1.5, -0.5])
         ax = sns.set(font_scale=2.0)
         ax = sns.heatmap(confusion_DT,annot=1,cmap="Spectral",xticklabels=['Fail','Pass'],yticklabels=['Fail','Pass'])
         plt.show()
         ax1=plot_confusion_matrix(dt_model,X_test,Y_test,cmap='Spectral',values_format='d',display_labels=['Fail','Pass'])
         print("Accuracy:",dt_model.score(X_test,Y_test))
         print("Precision:",precision_score(Y_test,y_pred_DT))
         print("Recall:",recall_score(Y_test,y_pred_DT))
         print("F1 Score:",f1_score(Y_test,y_pred_DT))
```

```
Predicted values:
array([1, 1, 1, 1, 1])
Confusion matrix
[[ 0 15]
 [ 0 35]]
Classification report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.70      1.00      0.82        35

    accuracy                           0.70        50
   macro avg       0.35      0.50      0.41        50
weighted avg       0.49      0.70      0.57        50
```


```
In [25]: ax1=plot_confusion_matrix(dt_model,X_test,Y_test,cmap='Spectral',values_format='d',display_labels=['Fail','Pass'])
         print("Accuracy:",dt_model.score(X_test,Y_test))
         print("Precision:",precision_score(Y_test,y_pred_DT))
         print("Recall:",recall_score(Y_test,y_pred_DT))
         print("F1 Score:",f1_score(Y_test,y_pred_DT))
```



```
Accuracy: 0.7
Precision: 0.7
Recall: 1.0
F1 Score: 0.82
```

3.7.6. Nearest Neighbors classifier

```
In [26]: from sklearn.neighbors import KNeighborsClassifier
         knn_model=KNeighborsClassifier(n_neighbors=3,metric='minkowski')
         knn_model.fit(X_train,Y_train)
```

```
In [27]: y_pred_KNN=knn_model.predict(X_test)
         print("Predicted values:")
         y_pred_KNN[0:5]
         print("Confusion matrix")
         confusion_KNN=confusion_matrix(Y_test,y_pred_KNN)
         print(confusion_KNN)
         print("Classification report")
         print(classification_report(Y_test,y_pred_KNN))
         sns.heatmap(confusion_KNN,annot=True,cmap="Spectral")
         plt.xlabel('Predicted Class',fontsize=15)
         plt.ylabel('True Class',fontsize=15)
         plt.title('Confusion Matrix',fontsize=15)
         ax = plt.axis([-0.5, 1.5, 1.5, -0.5])
         ax = sns.set(font_scale=2.0)
         ax = sns.heatmap(confusion_KNN,annot=1,cmap="Spectral",xticklabels=['Fail','Pass'],yticklabels=['Fail','Pass'])
         plt.show()
         ax2=plot_confusion_matrix(knn_model,X_test,Y_test,cmap='Spectral',values_format='d',display_labels=['Fail','Pass'])
         print("Accuracy:",knn_model.score(X_test,Y_test))
         print("Precision:",precision_score(Y_test,y_pred_KNN))
         print("Recall:",recall_score(Y_test,y_pred_KNN))
         print("F1 Score:",f1_score(Y_test,y_pred_KNN))
```

```
Predicted values:
array([1, 1, 1, 1, 1])
Confusion matrix
[[ 3 12]
 [ 2 33]]
Classification report
              precision    recall  f1-score   support

           0       0.60      0.20      0.30        15
           1       0.73      0.94      0.82        35

    accuracy                           0.72        50
   macro avg       0.67      0.57      0.56        50
weighted avg       0.69      0.72      0.65        50
```



```
Accuracy: 0.72
Precision: 0.73
Recall: 0.94
F1 Score: 0.82
```

3.8 Results

Logistic Regression: Acc: 0.71, Accuracy 0.68, Sensitivity 0.68, Specificity 0.68, Precision 0.87
N-Bayes: Acc: 0.72, Accuracy 0.72, Sensitivity 0.78, Specificity 0.55, Precision 0.75
DTree: Acc: 1.00, Accuracy 0.70, Sensitivity 1.0, Specificity 0.00, Precision 0.87

References

1. Dua, D., Karra, S. Results training non-pruned survival of patients with head-future from serum creatinine and ejection fraction UCI Machine Learning Repository [Data set]. Irvine, CA: University of California, School of Information and Computer Science. https://doi.org/10.24432/C5Z89R
2. Ahmad T, Munir A, Bhatti SH, Aftab M, Raza MA. Survival analysis of heart failure patients: a case study. PLoS ONE. 2017; 12(7):e0181001. https://doi.org/10.1371/journal.pone.0181001 https://www.nature.com/articles/srep10775
3. Johns Hopkins Rheumatology. Creatine Phosphokinase (CPK). https://www.hopkinsarthritis.org/rheum-disease/clinical-tests/creatine-phosphokinase-cpk/ https://www.ncbi.nlm.nih.gov/pubmed/8291
4. Epidemiology Kaplan Meier Estimate Measurement. Overview Video https://www.youtube.com/watch?v=kqXfJ1bnpOQ https://www.youtube.com/watch?v=ofme2o29ngU&feature=youtu.be&t=1m30s
5. Prognosis and Cardiovascular Disease - AHA Journals https://www.ahajournals.org/doi/full/10.1161/01.CIR.0000089927.70092.9B
6. Increase in Creatinine and Cardiovascular Risk in Patients ... - Remained Outdoor 31, 2020, from https://www.jacc.org/doi/abs/10.1016/j.jacc.2010.08.624
7. Creatine (2). What is a sodium blood test? https://www.healthline.com/health/sodium-blood
8. Serum Albumin, and Unfavorable Lymphocyte - MedCalc https://www.medcalc.org/manual/valuepage.php
9. Chi-Square Goodness of Fit Test - Statistics Solutions https://www.statisticssolutions.com/chi-square-goodness-of-fit-test/